ESCUELA INTERNACIONAL POSGRADOS

Máster Avanzado de Programación en Python para Hacking, BigData y Machine Learning

CERTIFICACIÓN PCAP

LECCIÓN 6

Módulos y paquetes, excepciones, generadores y cierres y procesado de ficheros

ÍNDICE

INTRODUCCIÓN

En esta última lección, vamos a ver diferentes temas como son los módulos y paquetes, las excepciones, los generadores y cierres, las funciones lambda y cómo procesar ficheros de texto y binarios, centrándonos principalmente en aquellos aspectos que suelen aparecer de forma común en el examen de la certificación PCAP.

OBJETIVOS

Al finalizar esta lección serás capaz de:

1. Conocer cómo importar un módulo y cómo utilizar los módulos y paquetes.
2. Conocer cómo capturar y lanzar excepciones y los diferentes tipos de excepciones.
3. Conocer cómo crear generadores, iteradores, cierres y funciones lambda y cómo utilizarlos.
4. Conocer cómo trabajar con ficheros de texto y binarios.

Módulos y Paquetes: ¿Qué es un módulo?

Módulo: Fichero que contiene definiciones y declaraciones de Python y que luego se puede importar y usar cuando sea necesario.

Permite dividir una pieza de SW en partes separadas pero cooperantes.

Consta de:

- **Nombre**: Necesario conocerlo para poderlo utilizar.
- **Entidades**: Funciones, variables, constantes, clases y objetos.

Importante

- Python proporciona una cantidad bastante grande de módulos que junto a las funciones integradas forman la **Biblioteca estándar de Python**.
- Para poderlo utilizar y utilizar sus entidades es necesario **importarlo**.
- Hay 3 maneras diferentes de importarlo.

Módulos y Paquetes: Primera forma de importarlo

Importante

- Podremos utilizar todas las entidades del módulo.
- No incluye las entidades del módulo en el namespace del código.
- **Namespace**: Espaco en el que existen algunos nombre y estos no estran en conflicto entre sí.

Módulos y Paquetes: Segunda forma de importarlo

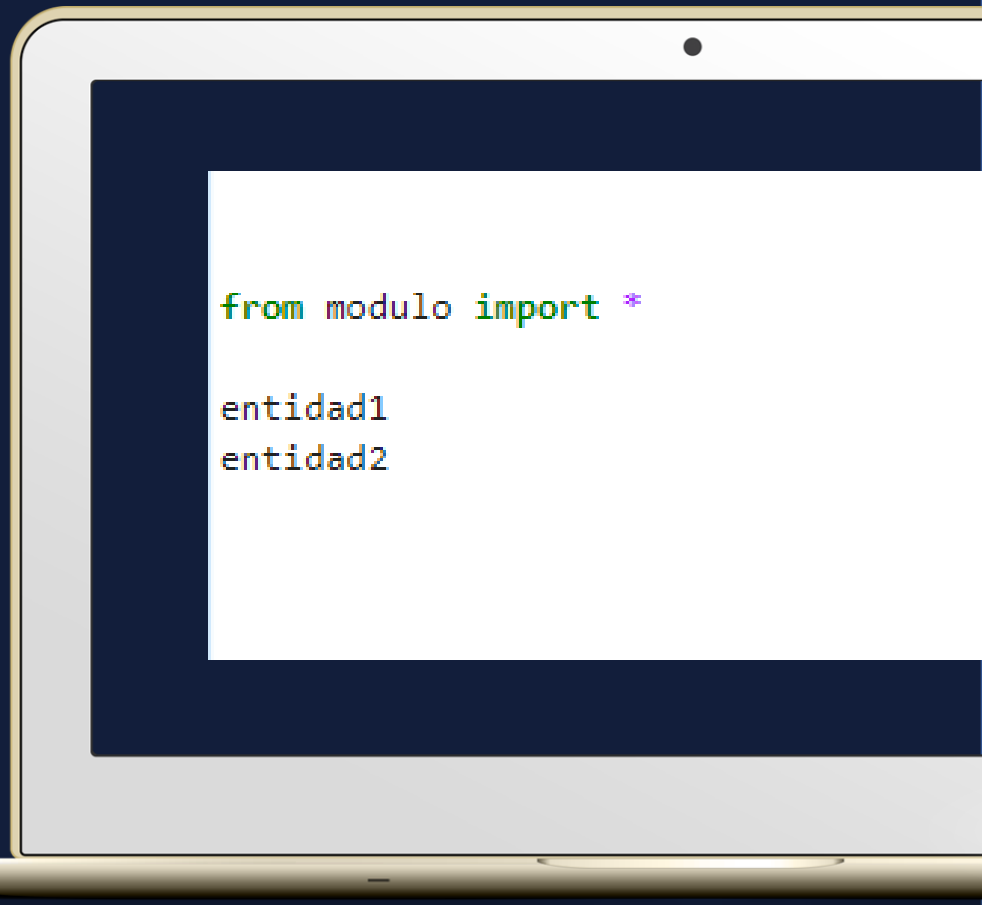
Importante

- Sólo se podrán utilizar las entidades enumeradas del módulo. El resto NameError.
- Incluye las entidades del módulo en el namespace del código.
- Posibles **conflictos** con mis entidades.

Módulos y Paquetes: Tercera forma de importarlo

Importante

- Se podrán utilizar todas las entidades del módulo.
- Incluye las entidades del módulo en el namespace del código.
- Posibles **conflictos** con mis entidades. Difícil controlarlos.
- Forma más agresiva y, por ello, no es recomendable utilizarla.



Módulos y Paquetes: Aliasing

Importante

- Con el alias se puede cambiar el nombre del módulo o entidad importada.
- Al cambiar el nombre del módulo o entidad ya no se podrá utilizar el original. NameError.

Módulos y Paquetes: Función dir()

Importante

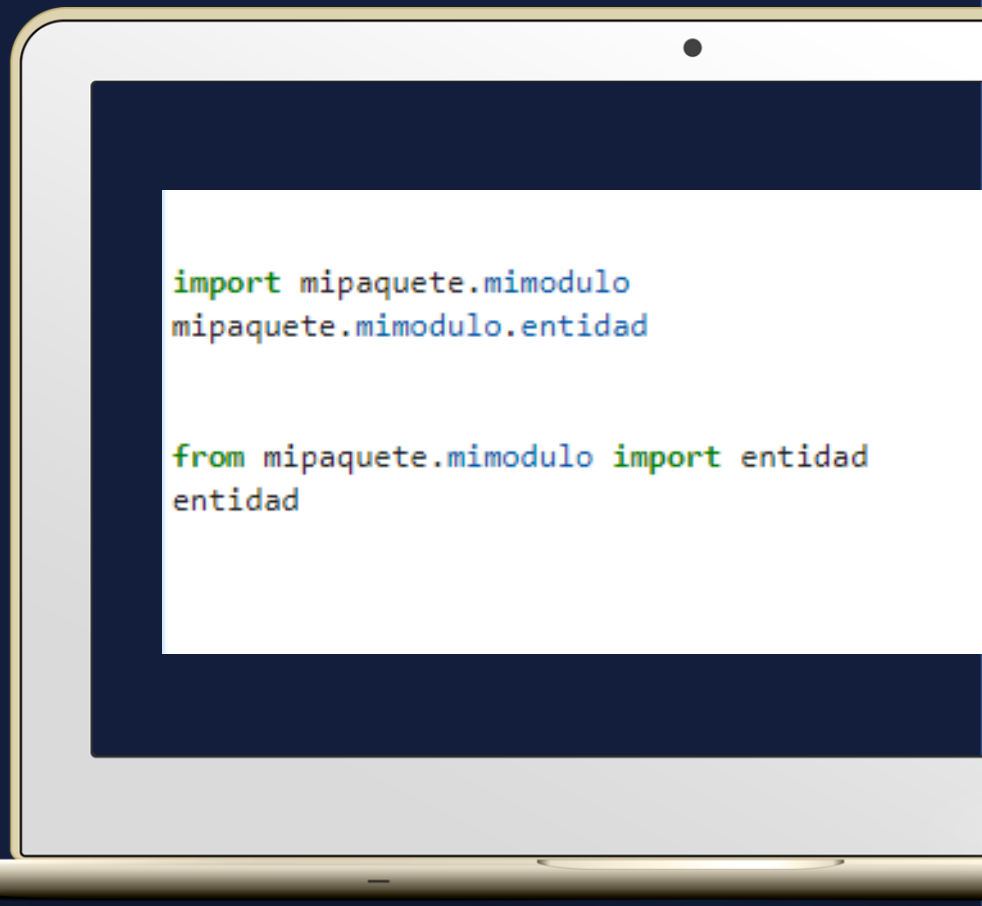
- La función dir(modulo) devuelve una lista ordenada alfabéticamente con todos los nombres de las entidades del módulo.
- dir() sólo se puede utilizar si el módulo se ha importado con la primera manera de importación vista.

Módulos y Paquetes: ¿Qué es un paquete?

Paquete: Permite agrupar los módulos

Importante

- Para crear un paquete se deben guardar los módulos en el mismo directorio y crear un fichero __init__.py.
- El fichero _init__.py puede llevar código para inicializar el paquete o estar vacío, pero debe existir.
- Subpaquetes: fichero __init__.py opcional.



Módulos y Paquetes: ¿Qué es un paquete?

Paquete: Permite agrupar los módulos.

Permite dividir una pieza de SW en partes separadas pero cooperantes.

Consta de:

- **Nombre**: Necesario conocerlo para poderlo utilizar.
- **Entidades**: Funciones, variables, constantes, clases y objetos.

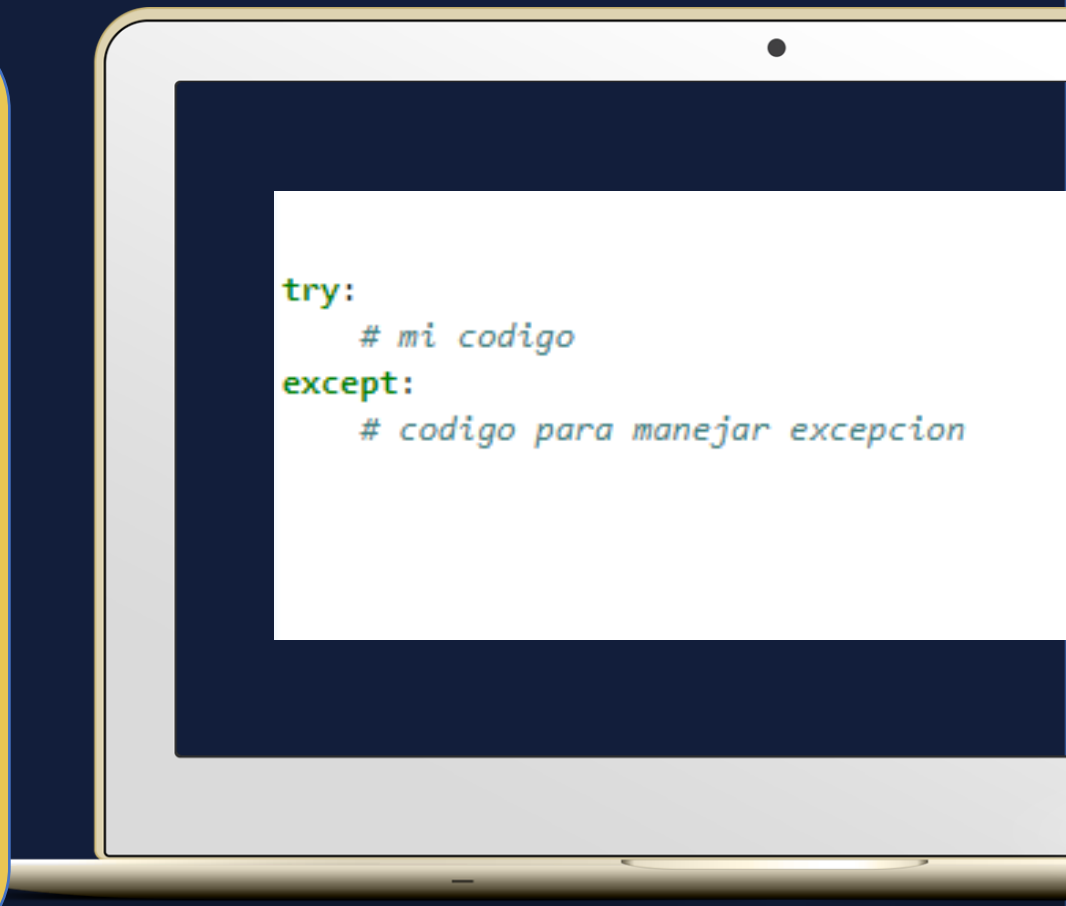
Importante

- Python proporciona una cantidad bastante grande de módulos que junto a las funciones integradas forman la **Biblioteca estándar de Python**.
- Para poderlo utilizar y utilizar sus entidades es necesario **importarlo**.
- Hay 3 maneras diferentes de importarlo.

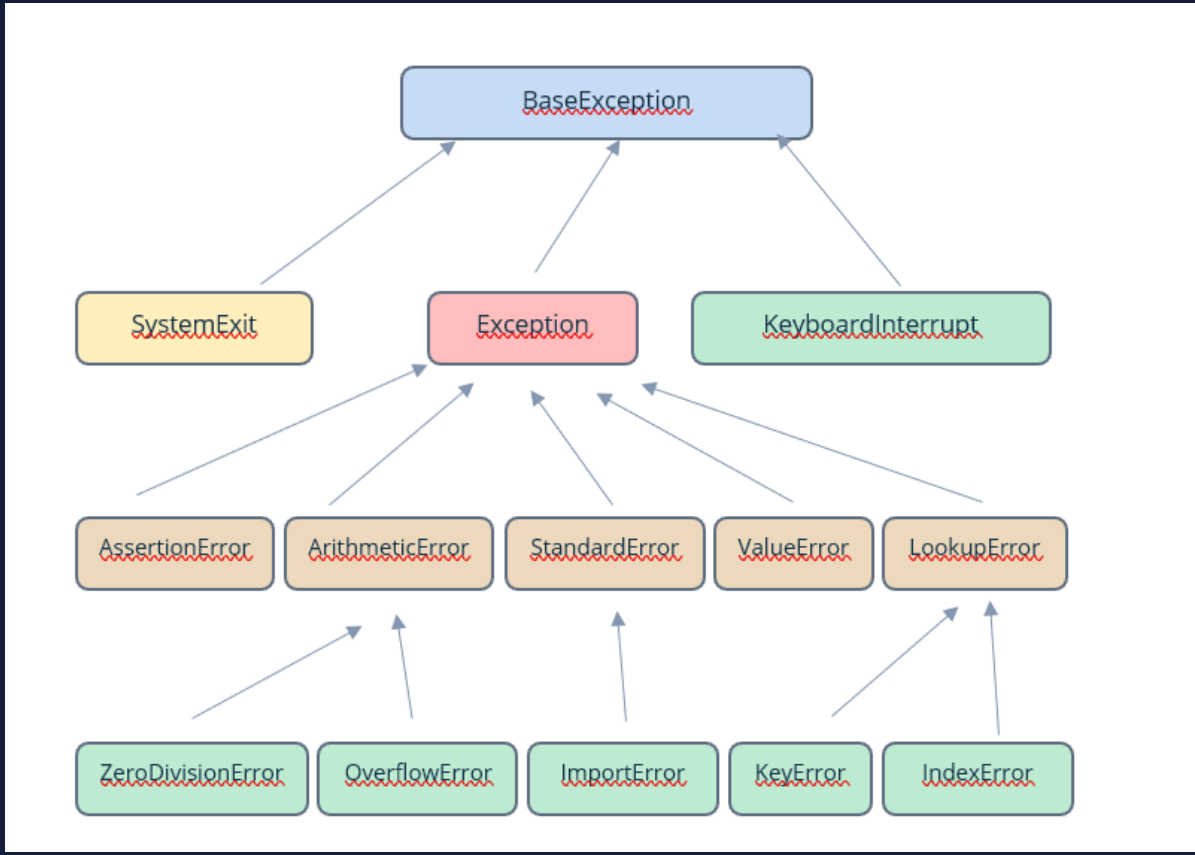
Excepciones: Control de errores

Importante

- **Lanzar excepción**: Cuando ocurre un error, Python detiene el programa y crea un tipo de dato Excepción.
- Una vez se lanza, Python espera a que se haga algo al respecto, si no se termina el programa de forma abrupta.
- Manejar la excepción: atender la excepción.
- El código dentro de except sólo se ejecuta si ha ocurrido un error.



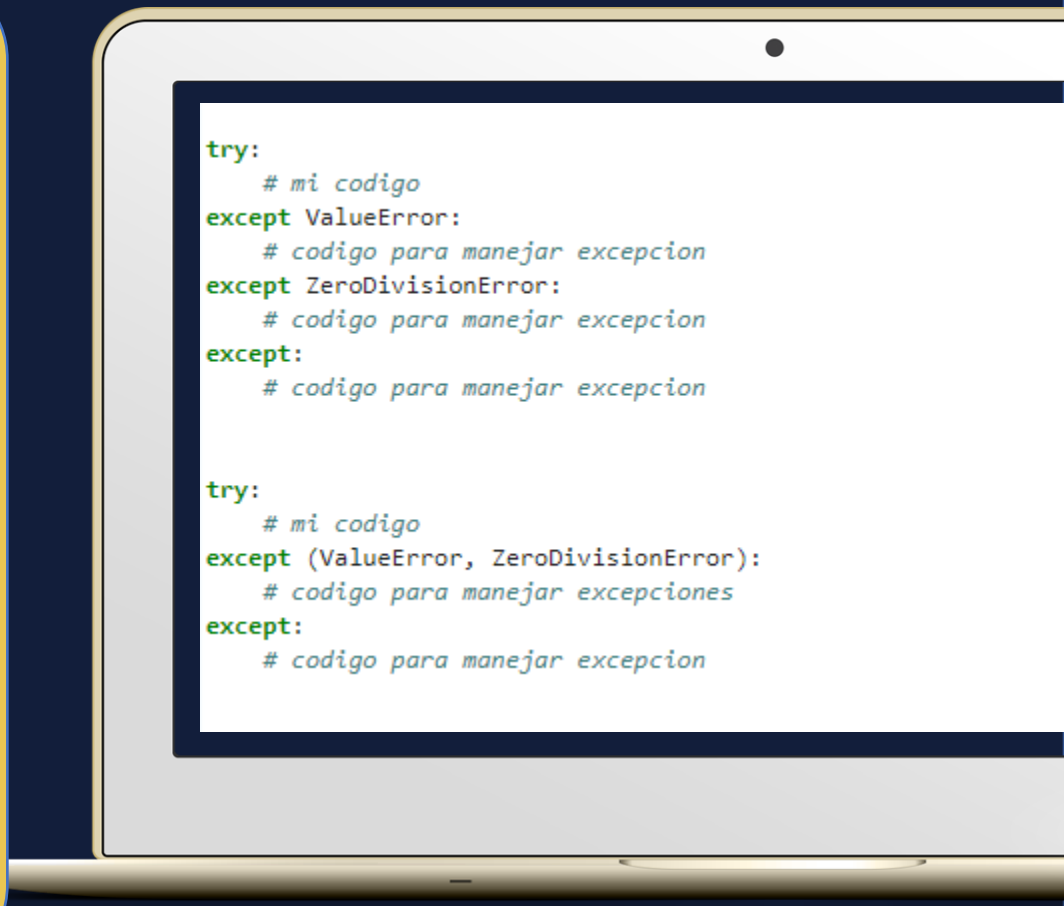
Excepciones: Jerarquía de clases



Excepciones: Capturar una excepción

Importante

- Bloques except analizados en mismo orden en que aparecen.
- Mínimo una cláusula except.
- Si uno de los except es ejecutado, ningún otro lo será.
- Si no coinciden con la excepción, la excepción permanece sin manejar.
- No poner excepciones más generales antes que otras más concretas.
- except sin nombre siempre al final.

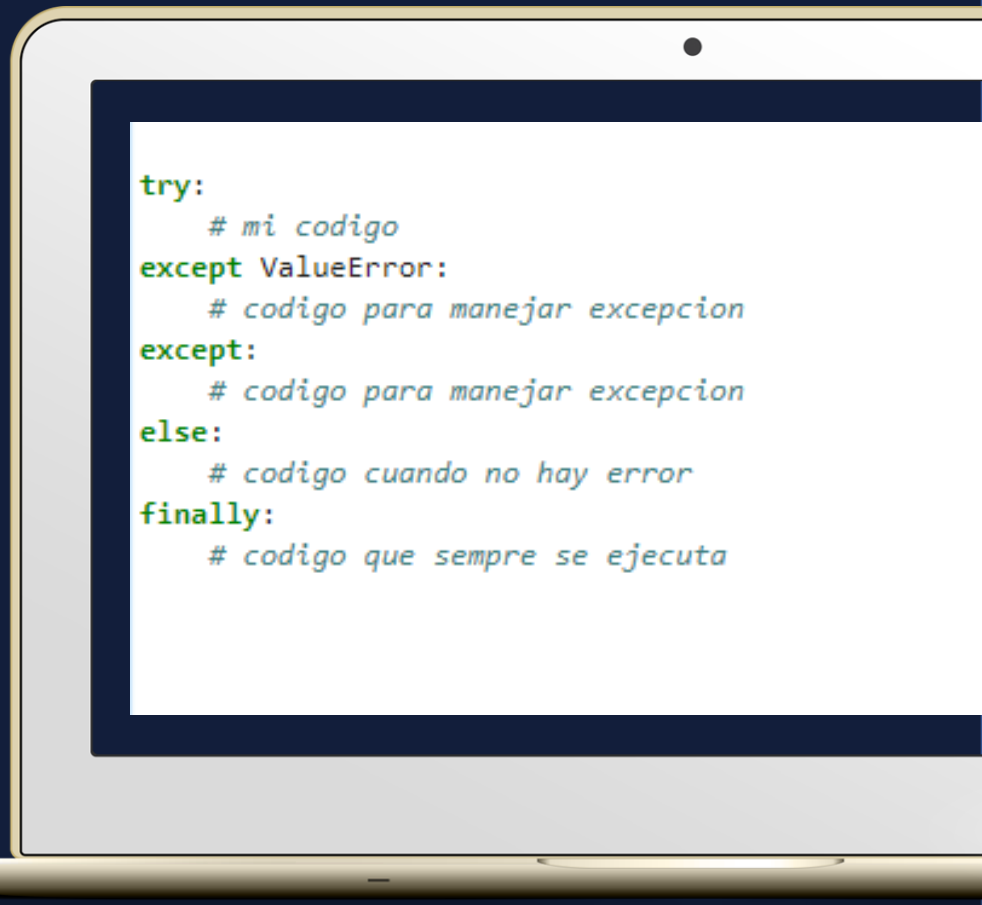


```
try:
    # mi codigo
except ValueError:
    # codigo para manejar excepcion
except ZeroDivisionError:
    # codigo para manejar excepcion
except:
    # codigo para manejar excepcion


try:
    # mi codigo
except (ValueError, ZeroDivisionError):
    # codigo para manejar excepciones
except:
    # codigo para manejar excepcion
```

Módulos y Paquetes: Capturar una excepción

Importante

- else sirve para definir un código que se ejecutará si no ha habido error.
- else debe ir después de los excepts.
- finally sirve para definir un código que siempre se ejecutará haya o no error.
- finally debe ir siempre al final.

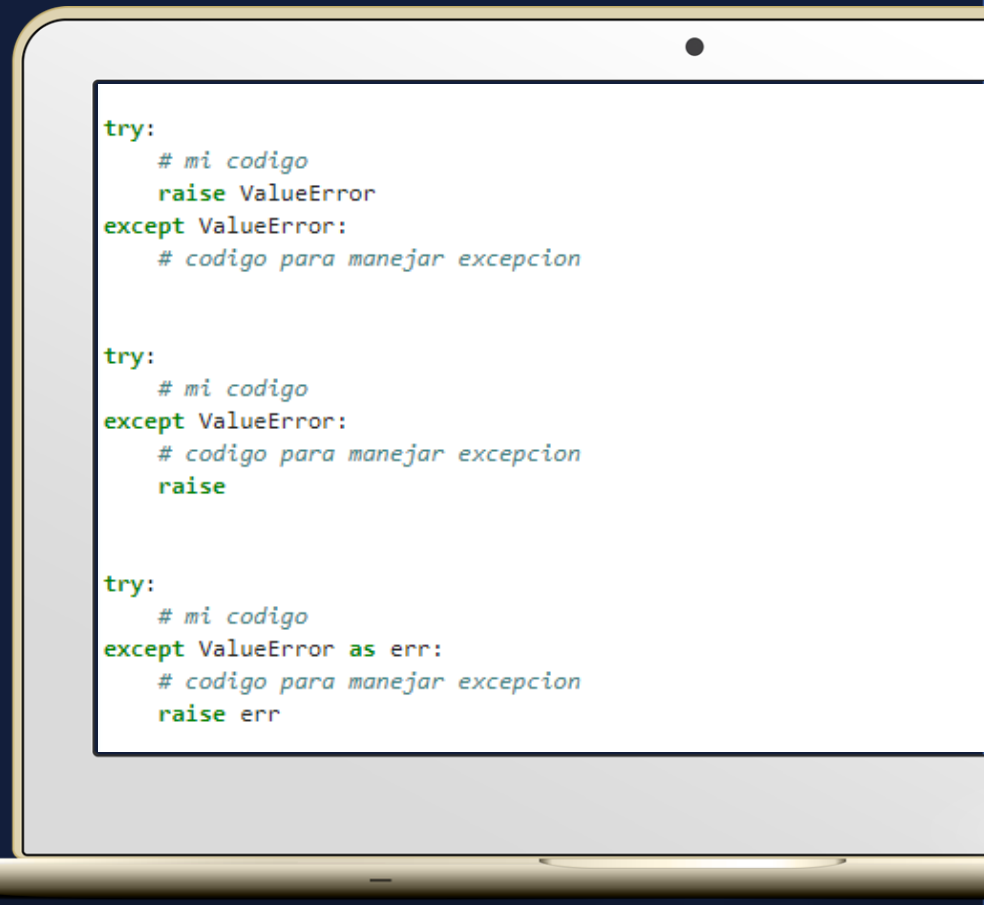


```
try:
    # mi codigo
except ValueError:
    # codigo para manejar excepcion
except:
    # codigo para manejar excepcion
else:
    # codigo cuando no hay error
finally:
    # codigo que sempre se ejecuta
```

Módulos y Paquetes: Lanzar una excepción

Importante

- Con raise se puede lanzar una excepción.
- Raise se puede utilizar en mi código o en la cláusula except.



```
try:
    # mi codigo
    raise ValueError
except ValueError:
    # codigo para manejar excepcion


try:
    # mi codigo
except ValueError:
    # codigo para manejar excepcion
    raise


try:
    # mi codigo
except ValueError as err:
    # codigo para manejar excepcion
    raise err
```

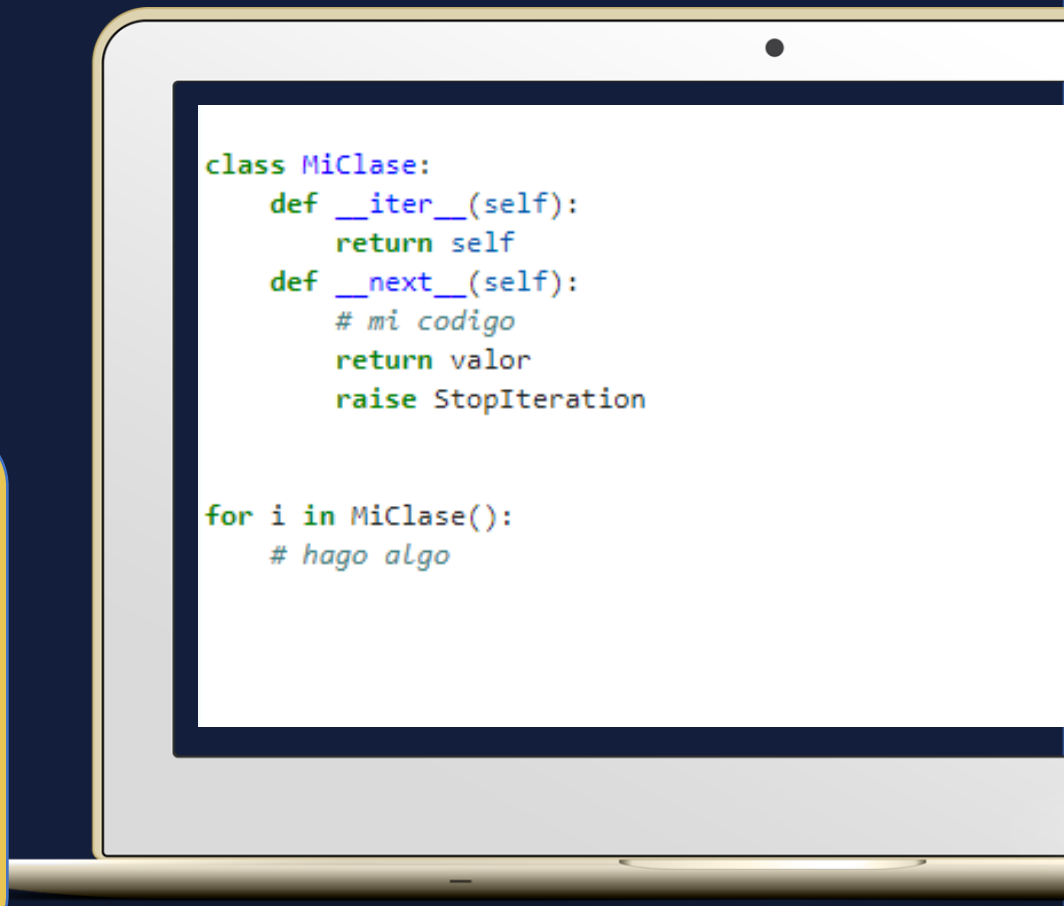
Generadores y Cierres: Generadores

Generador: Fragmento de código capaz de producir unaserie de valores y controlar el proceso de iteración. Denominados: iteradores.

Iterador: Objeto que cumple con el protocolo iterador.

Importante

- range() es un generador e iterador.
- __iter__ sólo se ejecutará una vez.
- La función __next__ se llamará hasta que lanza la excepción.



```
class MiClase:
    def __iter__(self):
        return self
    def __next__(self):
        # mi codigo
        return valor
        raise StopIteration


for i in MiClase():
    # hago algo
```

Generadores y Cierres: Cierres

Importante

- Al usar yield la función se convierte en generador y devuelve los elementos bajo demanda.
- Al llamar la función no devuelve los valores sino un objeto iterador.
- Cada vez que devuelve un calor no pierde el estado de la función.

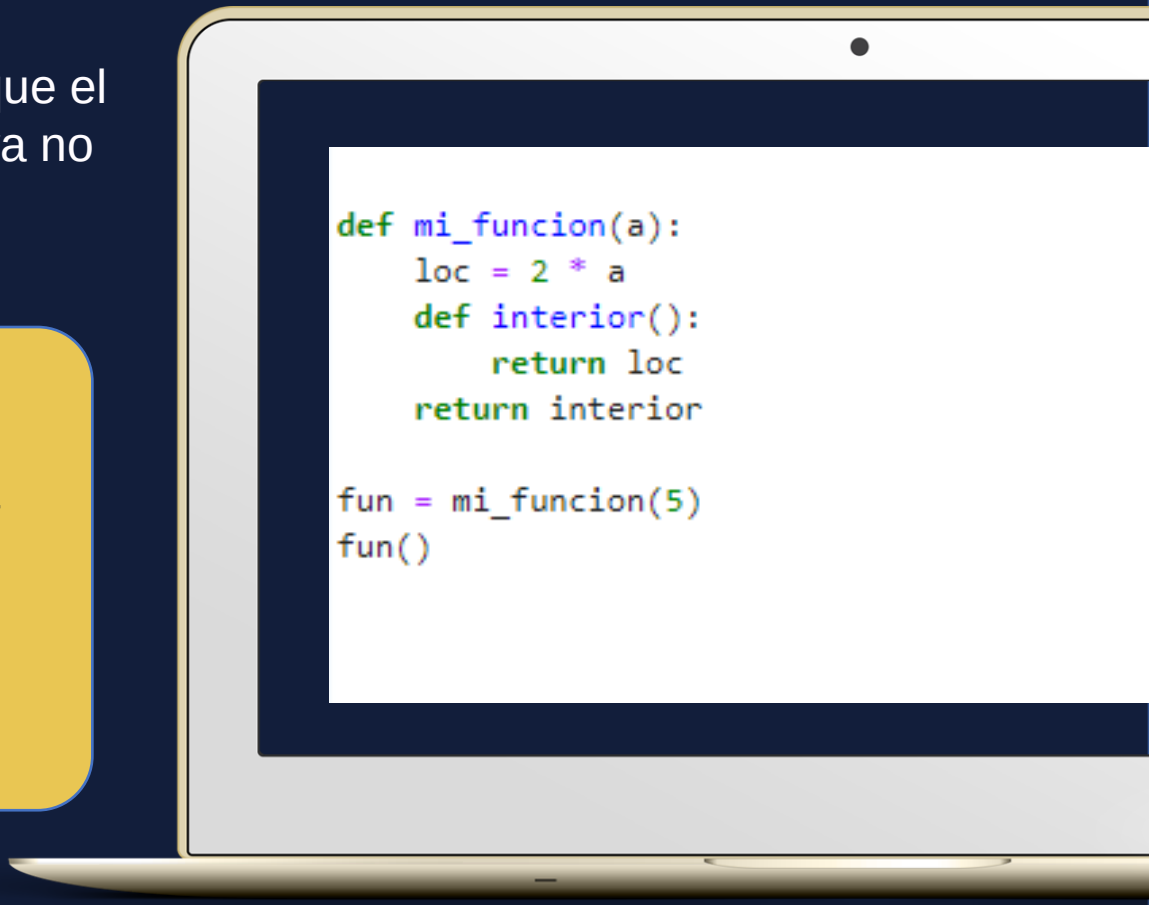
```
def mi_metodo(parametros):
    # mi codigo
    yield valor

for i in mi_metodo(parametros):
    # hacer algo
```

Generadores y Cierres: Generadores

Cierre: Técnica que permite almacenar valores a pesar de que el contexto en el que se crearon ya no existe.

Importante

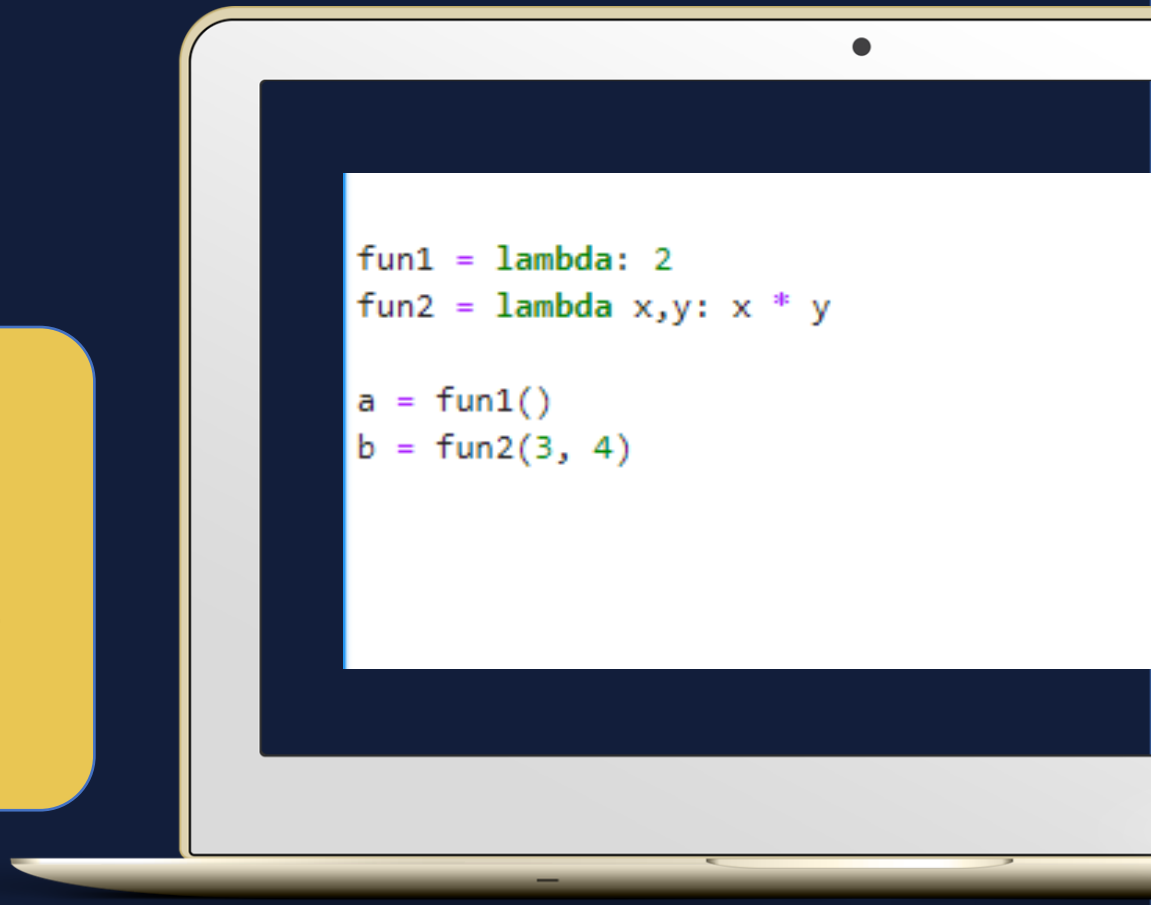
- Si la función interior puede tener parámetros, en ese caso al llamarla después habría que introducir el mismo número de parámetros.



```
def mi_funcion(a):
    loc = 2 * a
    def interior():
        return loc
    return interior

fun = mi_funcion(5)
fun()
```

Funciones lambda

Función lambda: Función sin nombre. Función anónima.



Importante

- Puede tener 0 o n argumentos.
- Siempre debe devolver un valor.

Funciones lambda: Funciones map() y filter()

map()

- Devuelve un iterador con los elementos de la lista pasada pasados por la función dada.

```
lista = list(map(lambda x: 2 * x, mi_lista))
```

filter()

- Devuelve un iterador con los elementos de la lista pasada que cumplen la función dada.

```
lista = list(filter(lambda x: x % 1 == 0, mi_lista))
```

Procesando archivos: Abriendo y cerrando el fichero

Stream: Entidad abstracta que se vincula al archivo físico para poder trabajar con él.

Importante

- La primera operación es la de abrir el archivo.
- La última operación es la de cerrar el archivo.

Procesando archivos: Modos de apertura

Modo de apertura	
r	Lectura. El fichero tiene que existir.
w	Escritura. El fichero puede existir o no. Si existe borrara contenido anterior.
a	Adjuntar. El fichero puede existir o no. Si existe escribirá al final del fichero.
r+	Leer y adjuntar. El fichero tiene que existir.
w+	Escribir y actualizar. El fichero puede existir o no. Si existe escribirá al inicio del fichero.
_t	El fichero es de tipo texto. Por defecto.
_b	El fichero es de tipo binario.
_x	Si el fichero ya existe lanza una excepción.

Procesando archivos: Leyendo ficheros de texto

Importante

- read() -> Lee todo el contenido del fichero y lo devuelve como una cadena de texto.
- read(num_chars) -> Lee el número de caracteres indicado. No hay más cadena vacía.
- readline() -> Lee de línea en línea. No hay más cadena vacía.
- readlines() -> Devuelve lista de líneas.
- stream es un objeto iterable. Se puede iterar para leerlo.

```
texto = stream.read()

linea = strem.read(num_caracteres)

linea = stream.readline()

lista_lineas = stream.readlines()

for linea in stream:
    # hacer algo
```

Procesando archivos: Escribiendo ficheros de texto

Importante

- write() -> Escribe la línea en el fichero.
- writelines() -> Escribe una lista de líneas.

Procesando archivos: Leyendo ficheros binarios

bytearray: Contenedor de bytes.

Importante

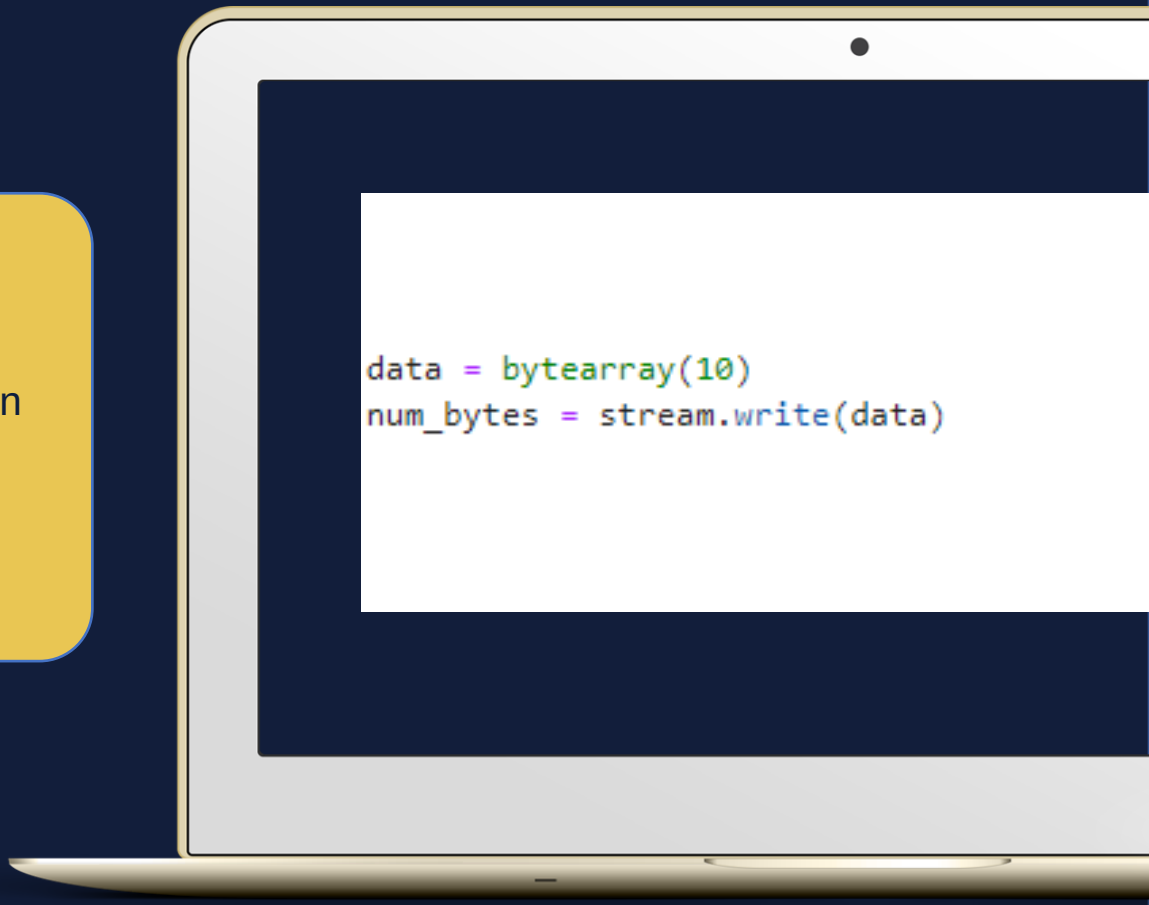
- bytearray -> Al crearlo, se rellena todo de 0s. Es mutable. Los elementos se pueden tratar como enteros.
- readinto() -> Lee el contenido del fichero y lo guarda en el bytearray. Devuelve el número de bytes leídos. Sólo lee hasta que llena el bytearray.
- read() -> Lee todo el contenido del fichero. También se le puede indicar el número de bytes a leer.

```
data = bytearray(10)
num_bytes = stream.readinto(data)

data = bytearray(stream.read())

data = bytearray(stream.read(num_bytes))
```

Procesando archivos: Escribiendo ficheros binarios

Importante

- write() -> Escribe el bytearray en el fichero.



CONCLUSIONES

1. Se han estudiado los módulos y paquetes, su importación y cómo trabajar con ellos.
2. Se ha estudiado cómo capturar y lanzar una excepción y su jerarquía e implicaciones.
3. Se ha estudiado qué son los generadores, iteradores y cierres, su utilidad y cómo crearlos y utilizarlos. Se han estudiado las funciones lambda y su utilización con map() y filter().
4. Se ha estudiado cómo trabajar con ficheros de texto y binarios.

MUCHAS GRACIAS POR SU ATENCIÓN

eip ESCUELA INTERNACIONAL POSGRADOS

tcivera@grupomainjobs.com

Tamara Civera Lorenzo
es.linkedin.com/in/tamara-civera-lorenzo-95962147

twitter.com/eiposgrados

facebook.com/eiposgrados

instagram.com/eiposgrados